Error Detection/Correction

Whenever digital data is transferred from one location to another, there is a probability for error due either to device failure or noise. There are numerous ways to handle errors at the system level. Some systems detect errors and request retransmission of data. In other systems retransmission may be impossible or prohibitively expensive. In such systems the receiving equipment must not only be able to detect, but also correct the error.

Both error detection and error correction rely on the transmission of redundant information. This requires additional bits of data and lowers the overall efficiency of transmission. In parallel systems additional wires, transmitters, and receivers are required, whereas serial transmission systems use additional time to transmit the redundant information. All these methods cannot completely eliminate errors, but as the percentage of redundant data bits increases, the probability of undetected or uncorrected errors decreases.

The simplest and most common method of dealing with errors is the addition of a single extra bit, called a parity bit. The parity bit is chosen such that the total number of ones in the word (counting the parity bit) is odd (in an odd parity system) or even (in an even parity system). Odd parity is generally preferred, since it insures at least one "1" in any word. At the receiving end, the parity of the word is examined. If any single bit in the word was changed, the detector indicates wrong parity. However, if an even number of errors occurs, this simple method cannot detect it. The parity bit provides only single error detecting.

Parity Generation

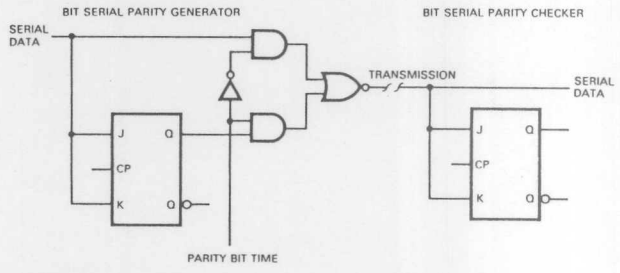

Figure 5-20

In a serial parity generator, a flip-flop is toggled for every "1" in the data word and the state of this flip-flop is inserted as a trailing parity bit. On the receiving side the parity checker has an equivalent flip-flop. Its state is interrogated after the data has been received. Both circuits are easily adapted for odd or even parity systems.

For parallel systems it is necessary to generate the modulo 2 sum of many inputs simultaneously. This requires an array of cascaded Exclusive-OR circuits. The 'F280 9-bit parity checker/generator is specifically designed for this function.

Error Correction[1]—Hamming Codes

A parity bit can only detect single errors. It cannot reliably detect multiple errors and it cannot correct single errors. A single redundant bit does not carry enough information to do so. However, it is possible to add more redundant information to the data, formulated such that errors are not only detected, but also corrected.

A data word containing an error-correcting field of redundant information is called a Hamming code. It uses a series of parity bits generated and arranged so that a unique set of parity errors results from an error in any given bit position. For example, three redundancy bits can have a total of eight different states. Since one of these states must indicate "no error," the other seven states can be used to locate an error in any one of seven transmitted bits. Three of the transmitted bits are the redundancy bits themselves, leaving four data bits in which an error can be uniquely detected, and also corrected. The coding of the parity bits is done conveniently so the pattern of parity errors is the binary address of the bit in error. In general, a Hamming code contains 2^m-1 bits, m of which are the Hamming or check bits, $2^m - m - 1$ are the data bits.

Total Bits	Hamming Bits	Data Bits
7	3	4
15	4	11
31	5	26

Thus three additional parity (Hamming) bits can provide single error correction for 4-bit data words. The seven bits are arranged in the following way:

$P_0P_1D_0P_2D_1D_2D_3$

where D_0, D_1, D_2, D_3 are the four data bits;

P_0 is odd parity over bits D_0, D_1, D_3;

P_1 is odd parity over bits D_0, D_2, D_3;

P_2 is odd parity over bits D_1, D_2, D_3.

At the receiving end the three parity bits are again generated from the data bits using an identical scheme. Then these three parity bits are compared with the three transmitted parity bits. If they all match, there was no single error. If they differ, the pattern of mismatches is interpreted as a binary address of the bit in error.

A practical system avoids the additional comparison and generates the error address (E_{0-2}) by including the received parity bits in the parity check:

E_0 is odd parity over bits P_0, D_0, D_1, D_3;
E_1 is odd parity over bits P_1, D_0, D_2, D_3;
E_2 is odd parity over bits P_2, D_1, D_2, D_3.

This Hamming code can detect and correct single errors, but it will fail on double errors—it would correct the wrong bit. However, if one more overall parity bit is added, it is also possible to detect (but not correct) double errors. When the receiver finds the overall parity check correct and the error address is zero, there was no error. If the overall parity check is wrong and the error address is not zero, there was a single error which can be corrected. However, if the overall parity check is correct, but the error address is not zero, then there was a non-correctable double error.

[1]For a detailed description of the theory behind and the applications of error correcting codes, see Peterson and Weldon, ERROR CORRECTING CODES, Second Edition, The MIT Press, Cambridge, MA, 1972.

Cyclic Checks for Error Detection

Error detection schemes using parity checks are well known. A parity check on a character is called "vertical" parity and a check on corresponding bits of every character in a message (data block) is called "longitudinal" parity. Used together, they provide a satisfactory checking scheme. The measure of protection provided is better than using vertical or longitudinal parity alone. However, the level of redundancy to achieve this protection is relatively high. For example, if there are x bytes in a message each consisting of seven data bits and one parity bit, the ratio of number of check bits to data bits is 1/7.

Another checking scheme exists called polynomial or cyclic coding that can be designed to perform with higher efficiencies than traditional parities. The level of protection achieved with a 16-bit cyclic check is probably satisfactory for most practical purposes. When used with a data block consisting of 7x data bits, ratio of check to data bits is only 16/7x. The ratio approaches a limit of zero as x increases. This high efficiency is inducing designers to incorporate cyclic check schemes in modern data communication and peripheral equipment such as tapes and discs. Theoretical knowledge necessary for cyclic check implementation has existed for several years. However, widespread use has begun only recently in designs using integrated circuits. Because it is relatively new, many designers do not have the needed exposure to cyclic schemes and tend to shy away.

This discussion is intended to familiarize uninitiated readers with the algebraic concepts required to design circuits for implementing cyclic check schemes. Not only are these concepts of value to the hardware designer, but also to the diagnostic programmer who must generate the code to check the implemented logic for validity and failures.

Polynomial Notation and Manipulation

A very convenient way of expressing a bit stream (message) consisting of K bits is to think of it as a polynomial in a dummy variable x with K terms. The bits of the message are the coefficients in the polynomial. Thus, if 100100011011 is the message, it may be written as:

$$M(x) = 1(x)^{11} + 0(x)^{10} + 0(x)^9 + 1(x)^8 + 0(x)^7 + 0(x)^6 + 0(x)^5 + 1(x)^4 + 1(x)^3 + 0(x)^2 + 1(x)^1 + 1(x)^0$$

or

$$M(x) = x^{11} + x^8 + x^4 + x^3 + x + 1$$

To compute the cyclic check on a message, another polynomial P(x), called a generating polynomial, is chosen. The degree "r" of the P(x) is such that it is greater than zero but less than the degree of M(x). Moreover, P(x) has a non-zero coefficient in the x^0 term. Thus it is clear that for a given message length, more than one generating polynomial of desired length can be specified. Fortunately, several accepted standard generating polynomials exist. Most common are CRC-16 and CRC-12 which were originally proposed for the IBM binary synchronous communications.

CRC-16 is a 16-bit check resulting from a generating polynomial $x^{16} + x^{15} + x^2 + 1$ and CRC-12 is a 12-bit check resulting from $x^{12} + x^{11} + x^3 + x^2 + x + 1$. Theory suggests that use of CRC-16 and CRC-12 will catch all messages with an odd number of errors, all with a single error burst of less than 16 or 12 bits respectively and most of the few messages with larger error bursts.

Cyclic check computation involves manipulating M(x) and P(x) using laws of ordinary algebra, except that modulo 2 arithmetic is used. Because modulo arithmetic yields the same result for addition and subtraction, it is necessary only to consider three operations involving polynomials—addition, multiplication and division.

Addition of two polynomials, $x^6 + x^5 + x^2 + 1$ and $x^5 + x^4 + x^3 + x^2$, yields $x^6 + x^4 + x^3 + 1$ as shown below:

$$\begin{array}{rl} x^6 + x^5 + 0 + 0 + x^2 + 0 + 1 &= 1100101 \\ x^5 + x^4 + x^3 + x^2 + 0 + 0 &= \ \ 111100 \\ \hline x^6 + 0 + x^4 + x^3 + 0 + 0 + 1 &= 1011001 \end{array}$$

Multiplication of two polynomials, $x^7 + x^6 + x^5 + x^2 + 1$ and $x + 1$, results in $x^8 + x^5 + x^3 + x^2 + x + 1$:

$$\begin{array}{rl} (x^7 + x^6 + x^5 + x^2 + 1)(x + 1) &= (11100101) \times 11 \\ x^8 + x^7 + x^6 + 0 + 0 + x^3 + 0 + x + 0 &= 111001010 \\ x^7 + x^6 + x^5 + 0 + 0 + x^2 + 0 + 1 &= 011100101 \\ \hline x^8 + 0 + 0 + x^5 + 0 + x^3 + x^2 + x + 1 &= 100101111 \end{array}$$

It is interesting to note that multiplication of a polynomial by x^m results in a shifted bit pattern which is identical to the original except for zeros in the lower m positions. For example:

$x^5(x^{11}+x^{10}+x^8+x^4+x^3+x+1) = x^{16}+x^{15}+x^{13}+x^9+x^8+x^6+x^5$
where $x^{11}+x^{10}+x^8+x^4+x^3+x+1 = 110100011011$
and $x^{16}+x^{15}+x^{13}+x^9+x^8+x^6+x^5 = 11010001101100000$

Dividing $x^{13}+x^{11}+x^{10}+x^7+x^4+x^3+x+1$ by $x^6+x^5+x^4+x^3+1$ results in a quotient of $(x^7+x^6+x^5+x^2+x+1)$ and a remainder of (x^4+x^2) as shown below. Practically, it might be easier to divide by longhand if the bit pattern is used rather than the polynomial.

$x^{13}+x^{11}+x^{10}+x^7+x^3+x+1 = 10110010011011$;
$x^6+x^5+x^4+x^3+1 = 1111001$

```
            11100111
          ------------------
11110011  10110010011011
          1111001
          -------
           1000000
           1111001
           -------
            1110010
            1111001
            -------
             1011110
             1111001
             -------
              1001111
              1111001
              -------
               1101101
               1111001
               -------
                 10100
```

Thus, $Q(x) = 11100111 = x^7+x^6+x^5+x^2+x+1$;
$R(x) = 10100 = x^4+x^2$

Cyclic Check-Computing Procedure

To compute a check on M(x), a generating polynomial P(x) is chosen as mentioned earlier. Steps involved in check computation are as follows:

a) Message polynomial M(x) is multiplied by x^r where r is the degree of P(x). As noted earlier, this process yields zeros in the lower r positions of M(x). These vacated positions are in preparation for the r check bits that will be appended to the message. Also note that this process does not alter the message bit pattern.

b) The result obtained from step (a) is divided by P(x). This gives a quotient Q(x) and a remainder R(x). The remainder will be r bits or less.

c) The quotient is discarded and the remainder is added to the result of step (a). The remainder is the check. The message with this remainder at the tail end constitutes the transmitted polynomial T(x).

The following example illustrates the computation procedure. Let $M(x) = x^{11}+x^{10}+x^8+x^4+x^3+x+1 = 110100011011$ and $P(x) = x^5+x^4+x^2+1 = 110101$. Thus, r = 5 and $x^rM(x) = x^{16}+x^{15}+x^{13}+x^9+x^8+x^6+x^5 = 11010001101100000$

$x^rM(x) = 11010001101100000$

P(x) = 110101

Carrying this division, Q(x) = 100001100111 and R(x) = 1011.

Transmitted message T(x) is obtained by adding R(x) to $x^rM(x)$:

$x^rM(x)$ = 11010001101100000
R(x) = 01011

T(x) = 11010001101101011

Data Check

Note that transmission occurs from left to right; data thus is unmodified and check bits follow at the end.

Data Validation at the Receiver

The transmitted polynomial arrives at the receiver modified or unmodified depending on whether transmission has encountered errors or not. Clearly, one of the ways by which the receiver can ensure data validity is to recompute the check bits on the message using the same generator polynomial and compare them with the received check bits. If they agree, it is assumed that received data is good.

Alternately, the receiver can divide the complete received polynomial by the same generator polynomial P(x). If there are no errors, it can be shown that this division results in zero remainder. This property can be easily verified by long division of T(x) = 11010001101101011 by P(x) = 110101. If the division results in a non-zero remainder, it can be assumed that T(x) has been modified by errors. This may be verified by introducing error and performing the division. The process of dropping and picking bits can be viewed as adding another polynomial E(x) (error polynomial) to T(x).

For example, if $T'(x) = 10010001101101011$ is received, instead of Tx), $T'(x) = T(x) \otimes E(x)$ can be written where $E(x) = 01010001101101011$. It follows then that if $T'(x)$ is exactly divisible by P(x), the receiver is blind and indicates no errors. This only happens if E(x) is exactly divisible by P(x). Knowing the characteristics of the transmission medium, it is advisable to choose such a generating polynomial that the probability of error patterns occurring that are divisible by P(x) is extremely low. The process of not detecting such errors is somewhat analogous to the erroneous validity indication in normal parity schemes where multiple bit errors may cancel each other's contribution to the check.

Basic Polynomial Divider

Consider longhand division of the polynomial $x^{16} + x^{15} + x^{13} + x^9 + x^8 + x^6 + x^5$, i.e., 11010001101100000, by another polynomial $x^5 + x^4 + x^2 + 1$, i.e., 110101.

```
          100001100111
        ________________________
110101  11010001101100000
        110101
        ________________________
            101101
            110101
            ________________________
              110001
              110101
              ________________________
                100000
                110101
                ________________________
                101010
                110101
                ________________________
                111110
                110101
                ________________________
                  1011
```

From this example, longhand division procedure can be summarized: align the most significant bits of the partial remainder and divisor, borrowing from the dividend as required. This implies aligning the divisor and dividend to start the division process. Then subtract the divisor from the partial product using modulo 2 arithmetic. When all bits in the dividend are processed, the result is the remainder.

Subtraction in modulo 2 of two bits is the same as performing an Exclusive-OR operation and alignment of bits suggests a shift operation. Consider two registers as shown in Figure 5-21.

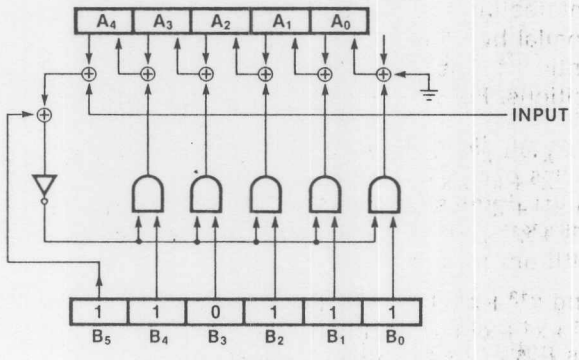

Figure 5-21 Conceptual Polynomial Divider

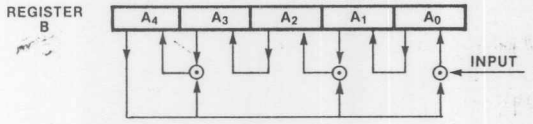

Figure 5-22 Polynomial Divider for $x^6 + x^5 + x^2 + 1$

Assume that register A is initially clear and register B contains 110101, which is the divisor bit pattern. Also imagine that the dividend serially enters the network as input (most significant bit first), in response to a clock signal that operates register A. As long as A_4 is cleared and B_5 is set, the AND gates are inhibited. This establishes a connection between A_4 input and A_3 output, A_3 input and A_2 output, etc. Thus, register A serves as a "shift left" register. When clocked with the dividend as serial input, the most significant bit eventually appears in A_4. At this point, A_4 and B_4 are both set, i.e., the most significant bits of divisor and dividend are aligned.

This alignment enables the AND gates. However, this has no effect on the Exclusive-OR gates with inputs derived from zero bit positions of register B. The "shift left" nature of register A at bit locations fed by these Exclusive-OR gates is preserved. Thus in Figure 5-21, the A_1 input comes from A_0, and A_3 input from A_2. On the other hand, the remaining bit positions receive the result of modulo 2 subtraction between appropriate bits. In summary, when register A is clocked after bit alignment, the partial remainder is loaded into it. If clocking is continued until all dividend bits are processed, the content of register A is the required remainder. Table 5-6 illustrates the register contents through this process; it is instructive to compare it with the long division.

Table 5-6
Bit Patterns Through Division Process

Input	Register				
	A_4	A_3	A_2	A_1	A_0
1	0	0	0	0	1
1	0	0	0	1	1
0	0	0	1	1	0
1	0	1	1	0	1
0	1	1	0	1	0
0	0	0	0	0	1
0	0	0	0	1	0
1	0	0	1	0	1
1	0	1	0	1	1
0	1	0	1	1	0
1	1	1	0	0	0
1	0	0	1	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	0	1	0	1
0	1	1	1	1	1
0	0	1	0	1	1

closer examination of Figure 5-21 suggests that it can be greatly simplified. Figure 5-22 shows a functionally identical scheme similar to that used for cyclic checking purposes.

Discussion of basic polynomial division circuits cannot be concluded without further observations. The division algorithm can be implemented by suitable interconnection of shift registers and Exclusive-OR gates. The total number of register positions equals the degree of the divisor polynomial. The total number of Exclusive-OR gates is equal to one less than the number of non-zero terms in the divisor.

Polynomial Divider for Cyclic Checks

But for one drawback, the polynomial divider could be used as a cyclic check generator. Imagine that the dividend polynomial $x^{16}+x^{15}+x^{13}+x^9+x^8+x^6+x^5$ is the result of multiplying $(x^{11}+x^{10}+x^8+x^4+x^3+x+1)$ by x^5, and the divisor $x^5+x^4+x^2+1$ is the generating polynomial. From the cyclic check coding scheme, remember that $x^{11}+x^{10}+x^8+x^4+x^3+x+1$ is the actual data stream. The divider circuit discussed so far does not provide the remainder until the trailing zeros have been processed. Thus, if the remainder is to be appended as a check to the data stream, there is a delay before it is available for transmission. In almost applications, such a gap between data and check bits is undesirable. This deficiency could easily be rectified if a circuit were possible which could multiply two polynomials while dividing by a third simultaneously.

Polynomial multiplication circuits can be derived using analogous arguments that result in the division circuit. For example, the arrangement shown in Figure 5-23 multiplies an incoming polynomial by $x^6+x^5+x^4+x^3+1$. Fortunately for cyclic check applications multiplication by a single term in the form x^r, where "r" is the degree of the generator polynomial, is sufficient. To implement a multiply by x^5 circuit, only a 5-bit shift register and one Exclusive-OR gate are needed as shown in Figure 5-24.

It is possible to combine the multiplier shown in Figure 5-24 and the divider in Figure 5-21 to implement a simultaneous multiply by x^5 and divide by $x^5+x^4+x^2+1$ circuit as shown in Figure 5-25. As before, Figure 5-25 may be simplified to arrive at Figure 5-26 which can be used as a cyclic check generator for the generating polynomial $P(x) = x^5+x^4+x^2+1$.

Table 5-7 lists the register content as each bit of the dividend (message polynomial) is processed.

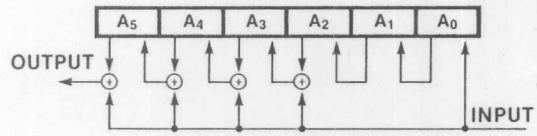

Figure 5-23 Circuit for Multiplying by $(x^6+x^5+x^4+x^3+1)$

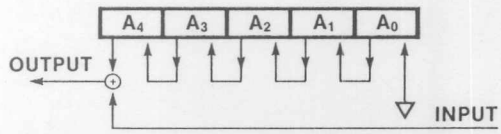

Figure 5-24 Circuit for Multiplying by x^5

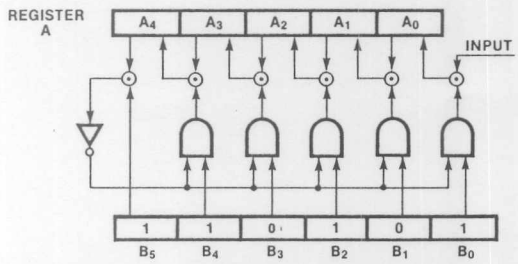

Figure 5-25 Conceptual Cyclic Check Generator

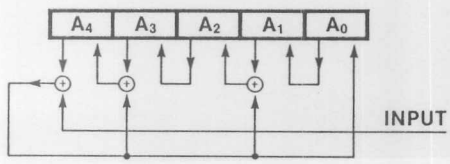

Figure 5-26 Basic Cyclic Check Circuit for P(x) = $x^5+x^4+x^2+1$

Table 5-7
Bit Pattern Through the Check Circuit

Input	Register				
	A_4	A_3	A_2	A_1	A_0
1	1	0	1	0	1
1	0	1	0	1	0
0	1	0	1	0	0
1	0	1	0	0	0
0	1	0	0	0	0
0	1	0	1	0	1
0	1	1	1	1	1
1	1	1	1	1	0
1	1	1	1	0	0
0	0	1	1	0	1
1	0	1	1	1	1
1	0	1	0	1	1

From Table 5-7, it is clear that the remainder is available as soon as the last data bit is processed. Also note that the quotient bit pattern appears in A_0. If it is desired to transmit the remainder from the register of Figure 5-26, it must be transmitted in a serial fashion. The connections must be established to make the register a straight shift from right to left by disabling the feedback through the Exclusive-OR gates.

Reverse Polynomials

Cyclic checks are often used in magnetic tape systems. Many of these have capabilities to read data in both forward and reverse directions. One of the reasons for this capability is to combat the overhead required to position the tape in front of the data block for a re-read operation in the event of an error. When "data followed by check bits" format is used to write on the tape, the check character is encountered first while reading in the opposite direction and the bit order for the whole block is reversed. Clearly, if the same check circuitry is used for error detection in both directions, erroneous indications are inevitable when reading in the opposite direction. This situation can be avoided by utilizing a reverse polynomial for checking in the opposite direction. The reverse polynomial is obtained by writing a polynomial bit pattern backwards. For example, the bit pattern for CRC-16 (forward) is 11000000000000101, i.e., $x^{16} + x^{15} + x^2 + 1$. The reverse polynomial for this pattern is 10100000000000011 or $x^{16} + x^{14} + x + 1$.

The 'F401 utilizes the concepts outlined above and contains polynomials for CRC-16, CRC-16 Reverse, CRC-12, LRC-8, CRC-CCITT and CRC-CCITT Reverse.

'F402 Expandable Polynomial Generator Checker

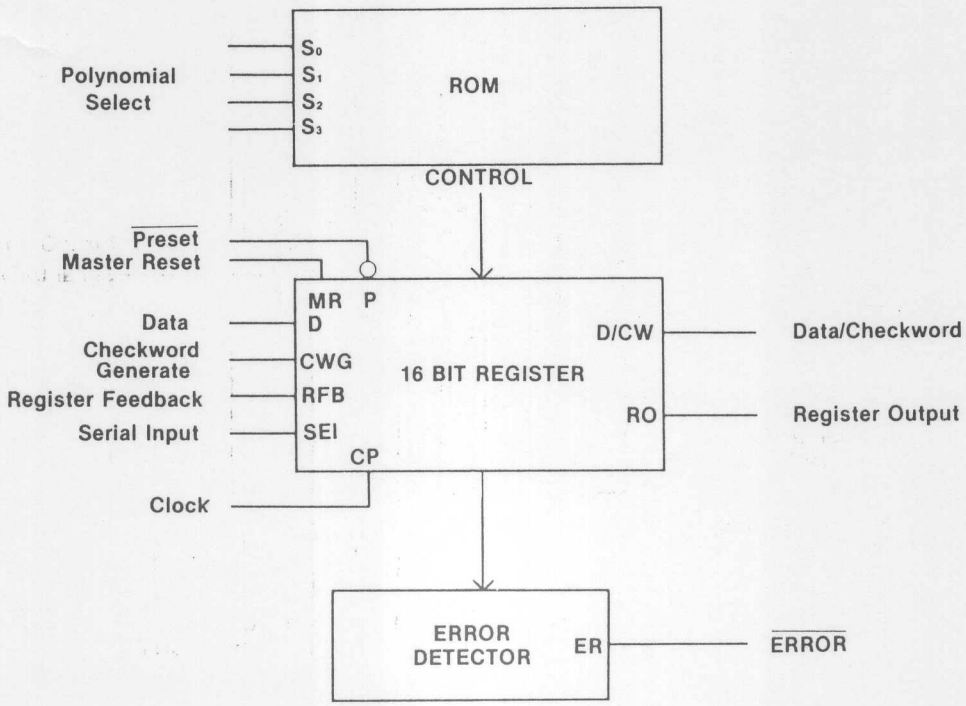

Figure 5-27

The 'F402 Expandable Polynomial Generator Checker is similar to the 'F401 and is expandable to polynomials of the 56th order. Six selectable polynomials and all data path gating is on the chip. The six polynomials contained on the chip are CRC16, CRC-CCITT, Ethernet, Ethernet Residue, 32nd Order, 48th Order and 56th Order. A bypass mode is also included in the selection to disconnect the chip from the feedback path. The 32nd, 48th and 56th order polynomials are all fire code generator polynomials; burst correction circuitry may be implemented with external circuitry.

'F402 for 16th Order Polynomials

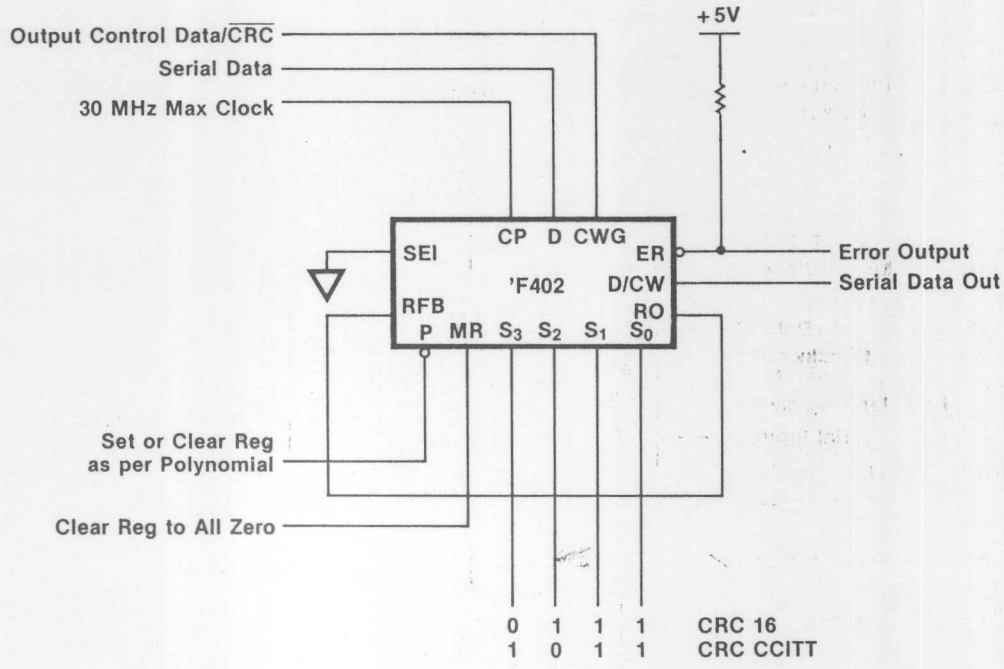

Figure 5-28

This shows the 'F402 in its simplest configuration, CRC16 or CRC-CCITT. The serial enable in line (SEI) is grounded, register out (RO) is connected to the register feedback (RFB), serial data in is applied to the data line (D), and transmitted data is taken from data/check word output (D/CW). The device can be clocked at 30MHz over temperature and voltage limits. Data or check word out is controlled by the check word generate line (CWG). The error flat ($\overline{ER}$) is an open collector output for expansion purposes.

'F402 for 32nd Order Polynomials

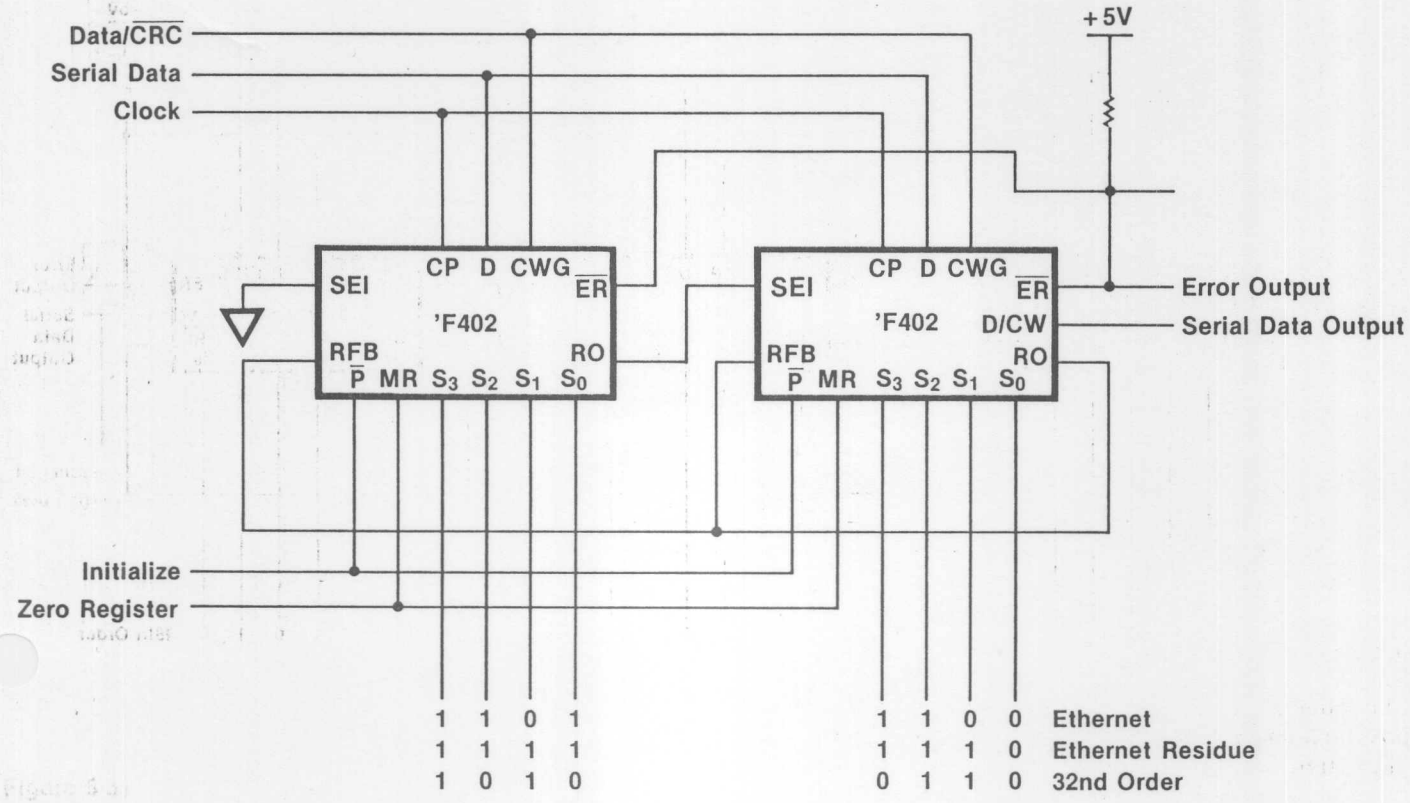

Figure 5-29

This diagram shows the ease of expansion of the 'F402. The register output (RO) of the most significant device is used to drive the register feedback (RFB) of both devices. The register output (RO) of the lower order device connects to the serial enable input (SEI) of the next higher order device; clock, data, check word generate and error flag lines of the respective devices are connected in parallel. A unique feature of the ethernet polynomial is that when the data stream and check word are divided by the polynomial, the result is not zero. To check for errors, the polynomial is changed to the ethernet residue polynomial and clocked once more. The error output will then go HIGH if no transmission error has occurred.

'F402 for 48th Order Polynomials

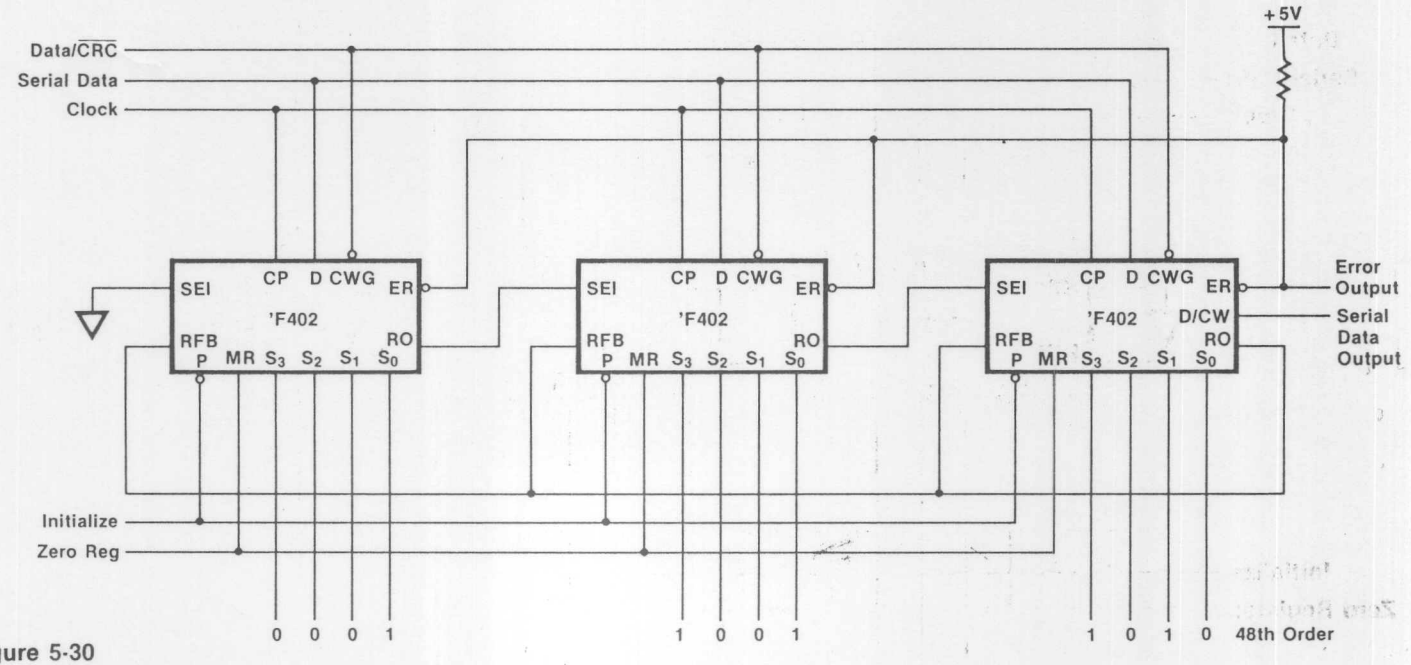

Figure 5-30

In this application another 'F402 is inserted into the feedback chain in a similar manner to the 32nd order configuration.

'F402 for 56th Order Polynomials and Lower

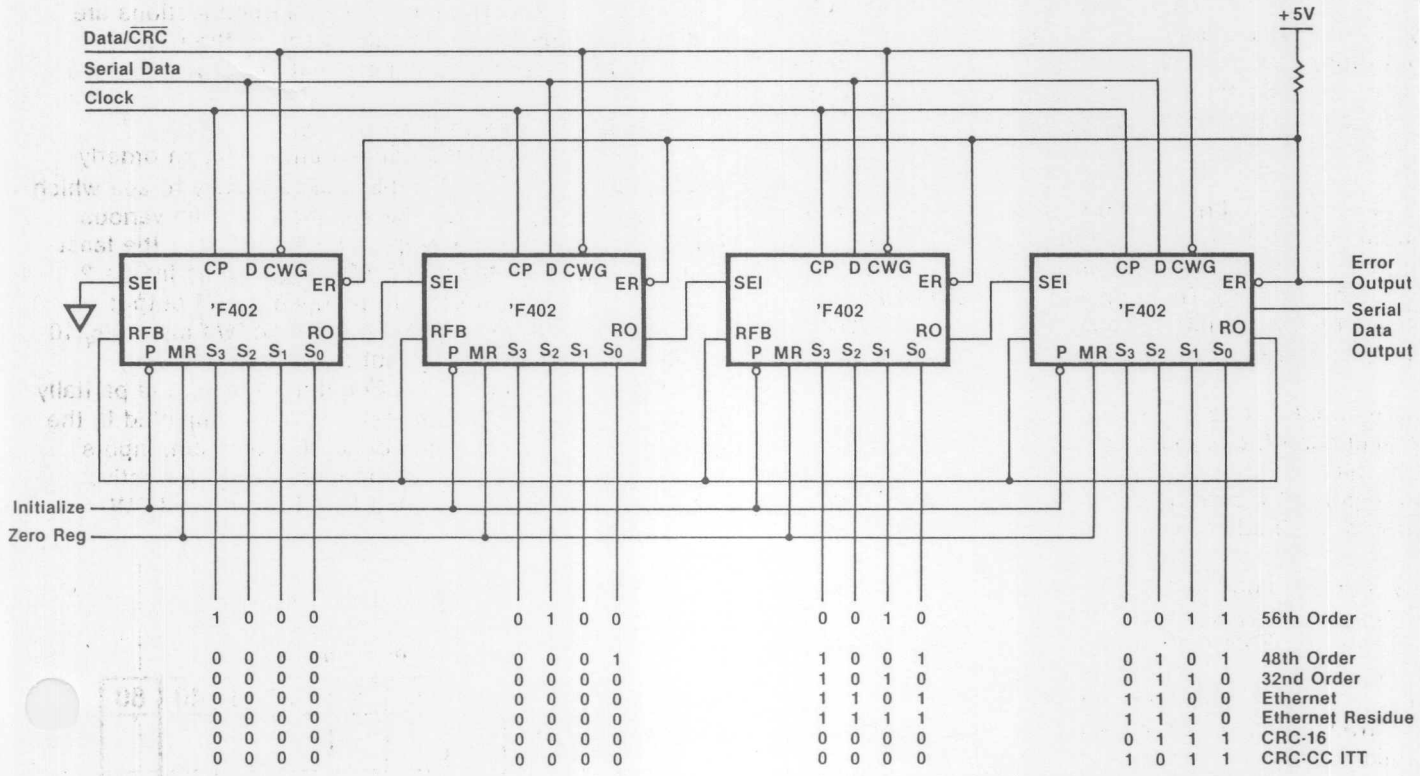

Figure 5-31

This shows a total of four 'F402s connected to form a 56th order polynomial generator checker. Note the 0000 select codes on the lower order registers when used with 48th, 32nd and 16th order polynomials.